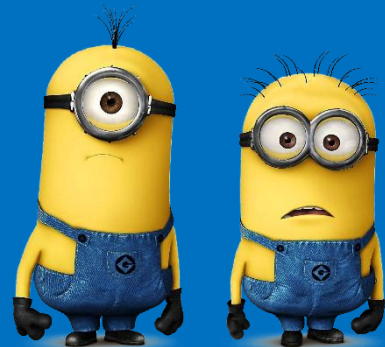




Programming
Languages

Лабораторная работа # 3 (13)

Python и ООП. Проектирование и реализация классов. Создание и манипулирование объектами



ЛАБОРАТОРНАЯ РАБОТА # 3 (13)

Python и ООП. Проектирование и реализация классов. Создание и манипулирование объектами

Цель работы

Изучить основы объектно-ориентированного программирования и приобрести *объектное мышление* для использования всей силы и мощи ООП на примере проектирования и реализации классов и объектов в Python.

Основное задание

Необходимо выбрать объекты реального мира и спроектировать классы (пользовательские типы данных) в Python для программного представления данных объектов (объект выбирается самостоятельно, но согласовывается с преподавателем). У объектов должно быть не менее 5 атрибутов (характеристик) и не менее трёх методов.

Написать программу для создания объектов спроектированных классов и демонстрации взаимодействия между ними.

Требования к выполнению

- 1) Программа должна обязательно быть снабжена комментариями на английском языке, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчика, номер группы и дату разработки.
- 2) Каждый класс должен иметь адекватное осмысленное имя (обычно это *имя существительное*) и начинаться с заглавной буквы. Имена полей и методов должны начинаться с маленькой буквы и быть также осмысленными (имя метода, который что-то вычисляет, обычно называют *глаголом*, а поле – именем существительным).
- 3) Каждый класс необходимо разместить в отдельном модуле, который затем подключается в другом модуле, где происходит создание объекта данного класса и его использование.

- 4) При проектировании классов необходимо придерживаться принципа единственной ответственности (Single Responsibility Principle), т.е. классы должны проектироваться и реализовываться таким образом, чтобы они были слабо связаны с другими классами при своей работе – они должны быть самодостаточными (стремитесь всегда разрабатывать масштабируемый и универсальный код).
- 5) Программа для демонстрации работоспособности разработанных классов должна быть снабжена дружелюбным и интуитивно понятным интерфейсом.
- 6) В отчёте **ОБЯЗАТЕЛЬНО** привести UML-диаграмму классов, которая демонстрирует классы и объекты приложения, их атрибуты и методы, а также взаимосвязь между ними.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)

Victor Ivanchenko



Что нужно запомнить (краткие тезисы)

1. **ООП** использует в качестве базовых элементов объекты, а не алгоритмы.
2. **Объект** – это понятие, абстракция или любой предмет с чётко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы. Введение объекта преследует две цели: *понимание прикладной задачи (проблемы)* и *введение основы для реализации на компьютере*.
3. **Главная идея ООП** – всё состоит из объектов! Программа, написанная с использованием ООП, состоит из множества объектов, и все эти объекты взаимодействуют между собой посредством посылке (передачи) сообщений друг другу. ООП и реальный мир не могут существовать отдельно!
4. **Программные объекты** могут представлять собой объекты реального мира или быть полностью абстрактными объектами, которые могут существовать только в рамках программы. **Гради Буч** (создатель унифицированного языка моделирования UML) даёт следующее определение объекта: «**Объект – это мыслимая или реальная сущность, обладающая характерным поведением, отличительными характеристиками и являющая важной в предметной области**».
5. **Каждый объект** имеет состояние, обладает некоторым хорошо определённым поведением и уникальной идентичностью.
6. **Состояние (state)** (синонимы: параметры, аспекты, характеристики, свойства, атрибуты, ...) – совокупный результат поведения объекта: одно из стабильных условий, в которых объект может существовать, охарактеризованных количественно; в любой конкретный момент времени состояние объекта включает в себя перечень параметров объекта и текущее значение этих параметров. Состояние объекта в Python реализуется с помощью описания полей класса.
7. **Поведение (behavior)** – действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния; видимая извне и воспроизводимая активность объекта. Поведение объекта в Python реализуется с помощью описания методов класса.
8. **Уникальность (identity)** – эта природа объекта; то, что отличает один объект от других. В машинном представлении уникальность объекта – это адрес размещения объекта в памяти. Следовательно, уникальность объекта состоит в

том, что всегда можно определить, указывают две ссылки на один и тот же объект, или на разные объекты.

9. Чтобы создать программный объект или группу объектов необходимо вначале описать где-то его(их) характеристики и шаблон поведения. Для этих целей в ООП существуют классы (*classes*). Формально, **класс** – это шаблон поведения объектов определённого типа с определёнными параметрами, которые описывают состояние объекта.
10. **Все экземпляры** (объекты) одного и того же класса имеют один и тот же набор характеристик (значение которых может быть различным у разных объектов) и общее поведение (все объекты одинаково реагируют на одинаковые сообщения).
11. **Каждый класс** может иметь также специальные методы, которые автоматически вызываются при создании и(или) уничтожении объектов класса:
 - **конструктор (*constructor*)** – служит для первоначальной инициализации состояния объекта и выполняется сразу же после создания объекта в памяти;
 - **деструктор (*destructor*)** – служит для освобождения всех ресурсов, которые были выделены для объекта, и выполняется перед полным удалением объекта из памяти сборщиком мусора (*garbage collector, GC*);
12. В упрощённом виде, класс – это кусок кода, у которого есть имя. Чтобы воспользоваться данным кодом, нужно создать объект (**экземпляр класса**).
13. **Инстанцирование** – процесс создания и инициализации объекта (экземпляра класса).
14. **Основные столпы (парадигмы) ООП** (порядок перечисления очень важен!):
 - 1) абстракция;
 - 2) инкапсуляция;
 - 3) наследование;
 - 4) полиморфизм;
 - 5) посылка сообщений (вызов соответствующих методов, свойств и т.д.);
 - 6) повторное использование кода.
15. **Абстрагирование в ООП** – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Абстракция – это набор всех таких характеристик.

16. **Абстракция в ООП** – это придание объекту характеристик, которые отличают его от всех других объектов, чётко определяя его концептуальные границы.

16. **Правила описания класса:**

- a. класс должен быть максимально простым, настолько простым, насколько это возможно;
- b. класс должен отвечать только за ту задачу, которая непосредственно на него возложена (класс должен описывать только одну группу объектов);
- c. название класса – очень важная вещь, оно должно быть простым и понятным (в идеале, имя класса должно говорить о том, за что он отвечает);
- d. название класса обычно называется именем существительным и записывается с заглавной буквы;
- e. классы должны быть самодостаточные (слабосвязанные), т.е. не зависеть от реализации других классов.

17. **Правила описания полей:**

- a. название поля должно однозначно отражать его содержимое;
- b. название поля обычно называется именем существительным и записывается с маленькой буквы;
- c. доступ к полю обычно должно осуществляться через интерфейсную часть объекта или класса.

18. **Правила описания методов:**

- a. название метода должно однозначно отражать, что он выполняет;
- b. название метода обычно называется глаголом и записывается с маленькой буквы;
- c. содержимое метода должно помещаться на экран монитора;
- d. один метод – одно действие!!!**

19. ООП помогает справиться со следующими проблемами:

- ✓ уменьшение **сложности** программного обеспечения (ПО);
- ✓ увеличение **производительности** труда программистов и **скорости** разработки ПО;
- ✓ повышение **надёжности** ПО;
- ✓ обеспечение возможности **модификации** отдельных компонентов ПО без изменения остальных его частей;
- ✓ обеспечение возможности **повторного использования** отдельных компонентов ПО.

Пример выполнения задания

Предметной (проблемной) областью является выставление оценок студентам преподавателем в университете и подсчёт успеваемости всех студентов.

Пункты решения поставленной задачи:

- 1) Выделим из предметной области все существительные и глаголы (действия), проанализируем их и попытаемся на базе них описать классы-сущности и функциональные классы с выделением в них существенных характеристик с точки зрения нашей будущей системы. Получаем:
 - ✓ существительные: оценка, преподаватель, университет, успеваемость, студент
 - ✓ действия: выставять оценки, подсчитывать успеваемость
- 2) После анализа предметной области спроектируем UML-диаграмму классов и их взаимосвязь друг с другом (смотрите рисунок 1):
 - ✓ студент (*Student*) – класс для описания студента и его успеваемости (класс-сущность, на базе которого будут создаваться далее бизнес объекты, а с этими объектами уже будет работать основная бизнес логика приложения);
 - ✓ преподаватель (*Teacher*) – класс для описания состояния и поведения объекта преподаватель (общий класс, в котором сосредоточена некоторая логика выставления оценок студентам и характеристики преподавателя (имя, стаж работы, и т.д.);
 - ✓ управляющий (*University Manager*) – класс, в котором описывается основная бизнес логика приложения – вычисление успеваемости студентов (функциональный класс);
 - ✓ Всевышний (*God*) – утилитный (дополнительный) класс для создания списка студентов, а также представления данного списка в строковом варианте для вывода на консоль (класс необязательный для доменной области, но облегчает её тестирование).

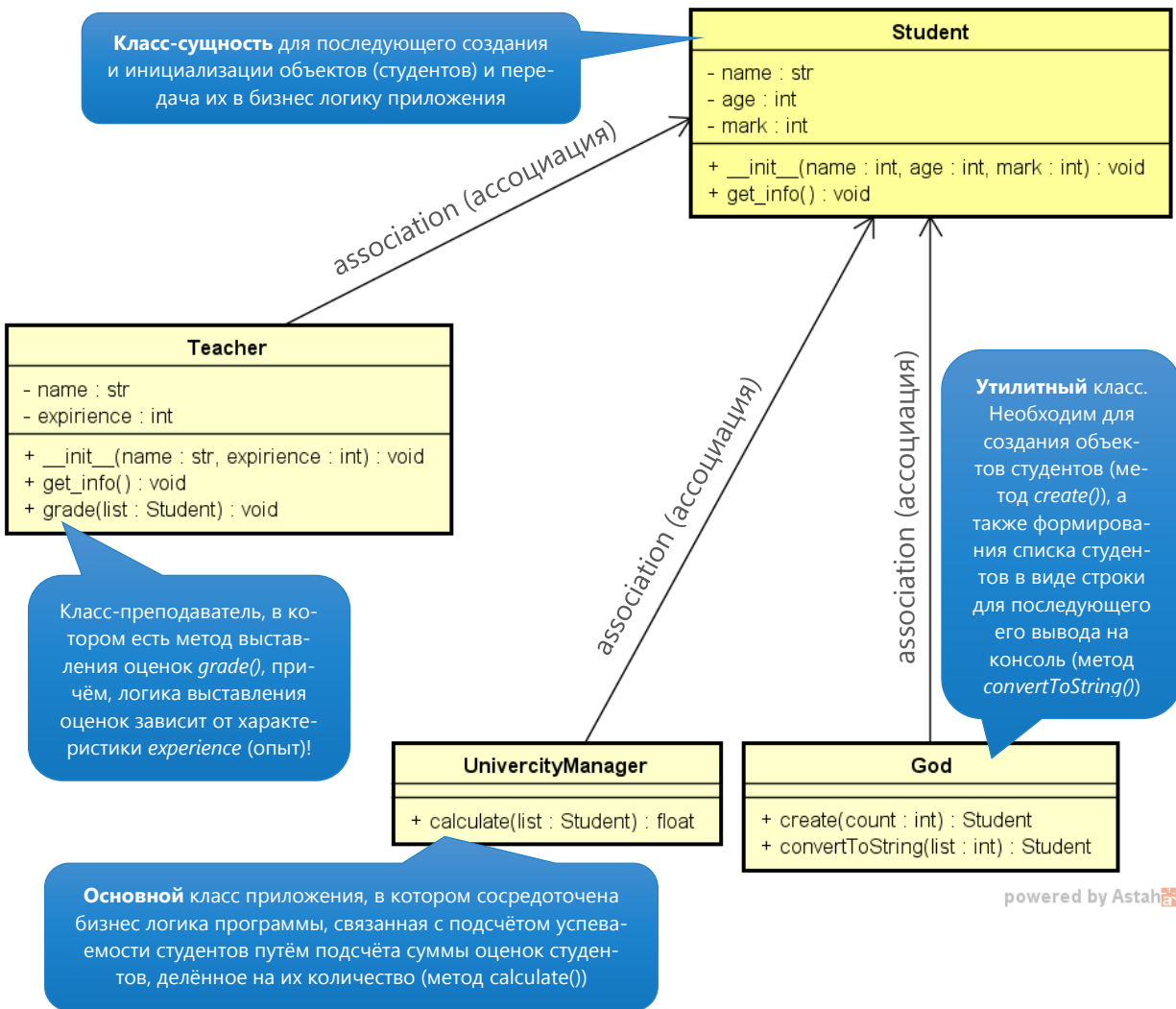


Рисунок 1 – UML-диаграмма классов заданной предметной (проблемной) области

- 3) На базе приведённой UML-диаграммы разработаем соответствующие классы проблемной области, причём код описания соответствующих классов разместим в отдельных одноимённых модулях (код исходных классов приведён на рисунках 2 - 5):


```

class Student:
    """ class defines student's information """

    def __init__(self, name, age, mark=4):
        self.name = name
        self.age = age
        self.mark = mark

    def get_info(self):
        return (self.name +
                "(age = " + str(self.age) +
                ", mark = " + str(self.mark) + ")")

```

Специальный метод класса – **конструктор**, который служит для первоначальной инициализации параметров (состояния) создаваемого объекта

Динамический метод для вывода строковых данных о текущем состоянии объекта

Рисунок 2 – Исходный код класса-сущности *Student*

```

import random
from student import Student

class Teacher:
    """class define teacher's information and logic"""

    def __init__(self, name, experience=1):
        self.name = name
        self.experience = experience

    def get_info(self):
        return (self.name + "("
                + str(self.experience) + ")")

    def grade(self, list_of_student):
        MIN_MARK = 4
        MAX_MARK = (100 - self.experience) // 10
        for st in list_of_student:
            if isinstance(st, Student):
                st.mark = random.randint(MIN_MARK, MAX_MARK)

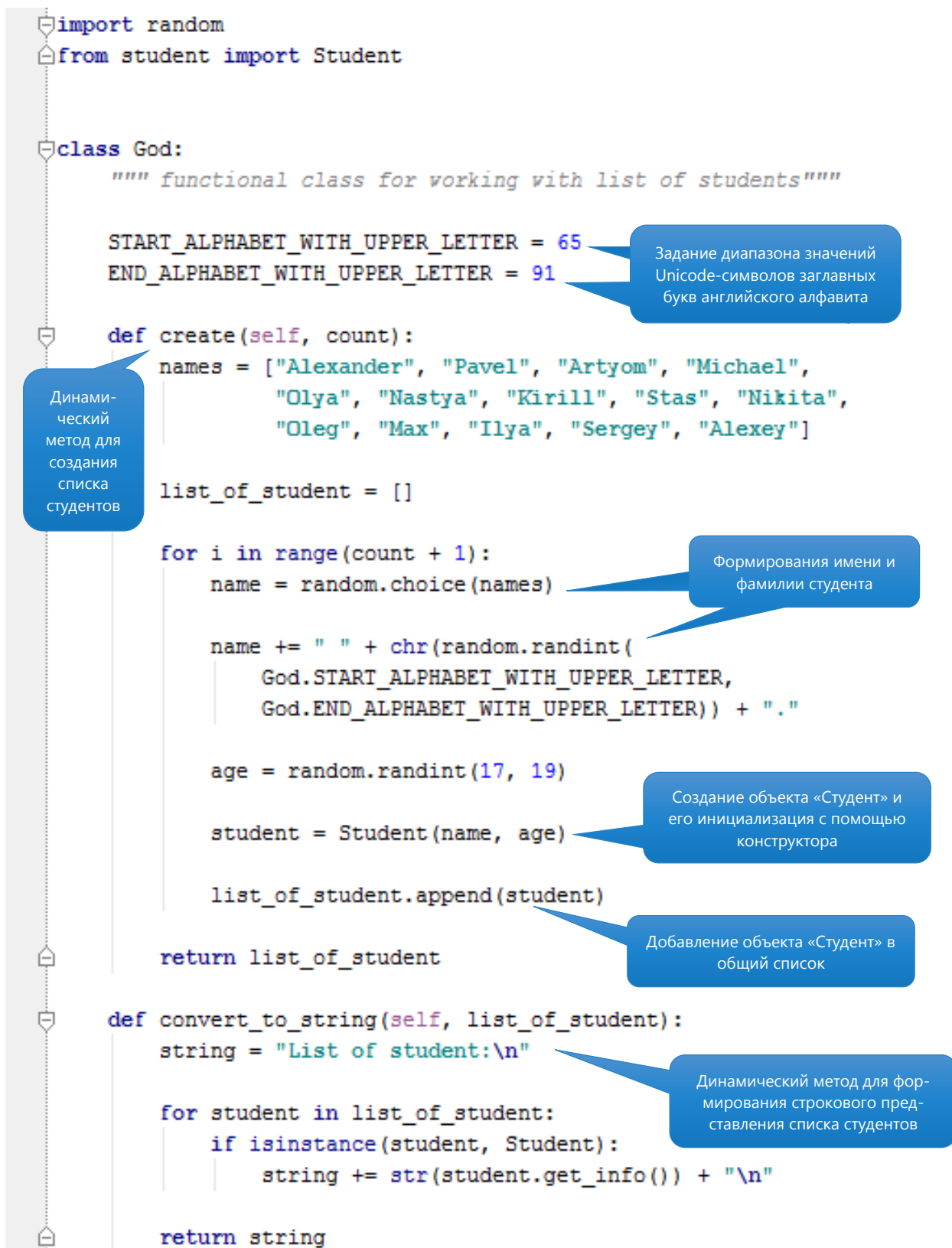
```

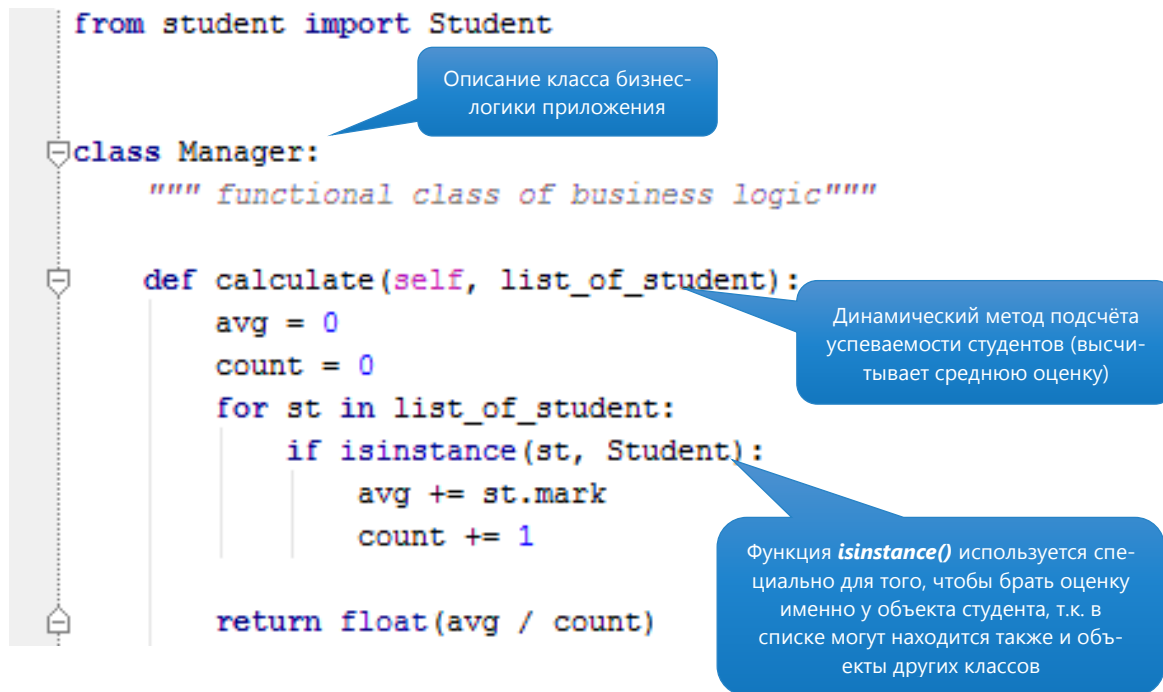
Импорт соответствующих модулей. Обратите внимание на порядок их описания: вначале идут всегда импортирование всего модуля, а затем только импортирование конкретных компонентов!

Конструктор класса Teacher для первоначальной инициализации параметров (состояния) создаваемого объекта

Динамический метод для представления оценок студентам. Обратите внимание, обычно все динамические методы хоть как-то используют состояние объекта для своей работы!

Рисунок 3 – Исходный код общего класса *Teacher*

Рисунок 4 – Исходный код утилитного (вспомогательного) класса *God*

Рисунок 5 – Исходный код функционального класса *Manager*

- 4) Для тестирования разработанной модели поведения системы создадим ещё один модуль `main` (см. рис. 6). В нём опишем функцию `main()`, в которой смоделируем создание списка студентов, выставление оценок преподавателем и подсчёт успеваемости студентов, а также вывода списка и результата подсчёта успеваемости в консоль (терминал).
- 5) Результат работы разработанной программной системы представлен на рисунке 7.

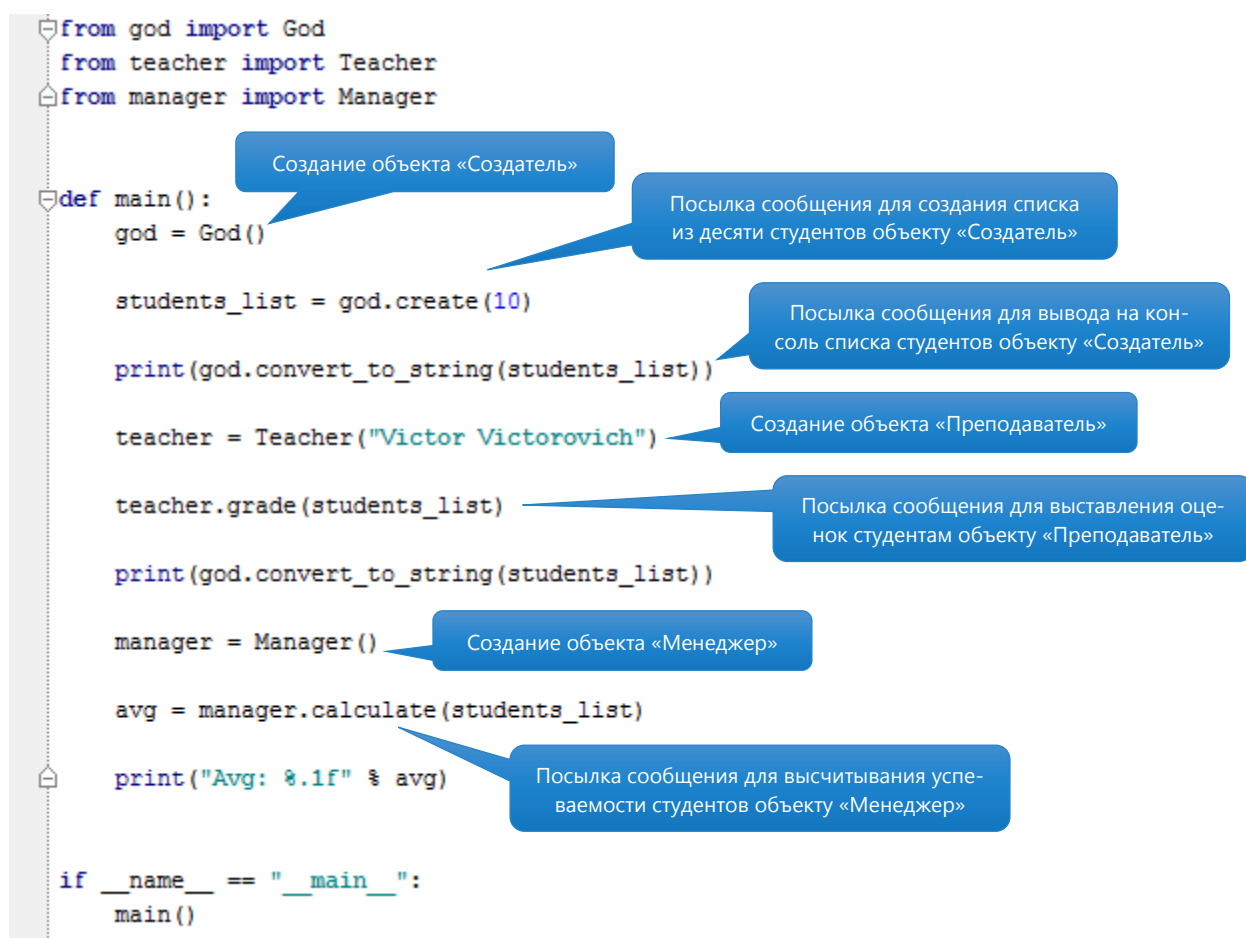
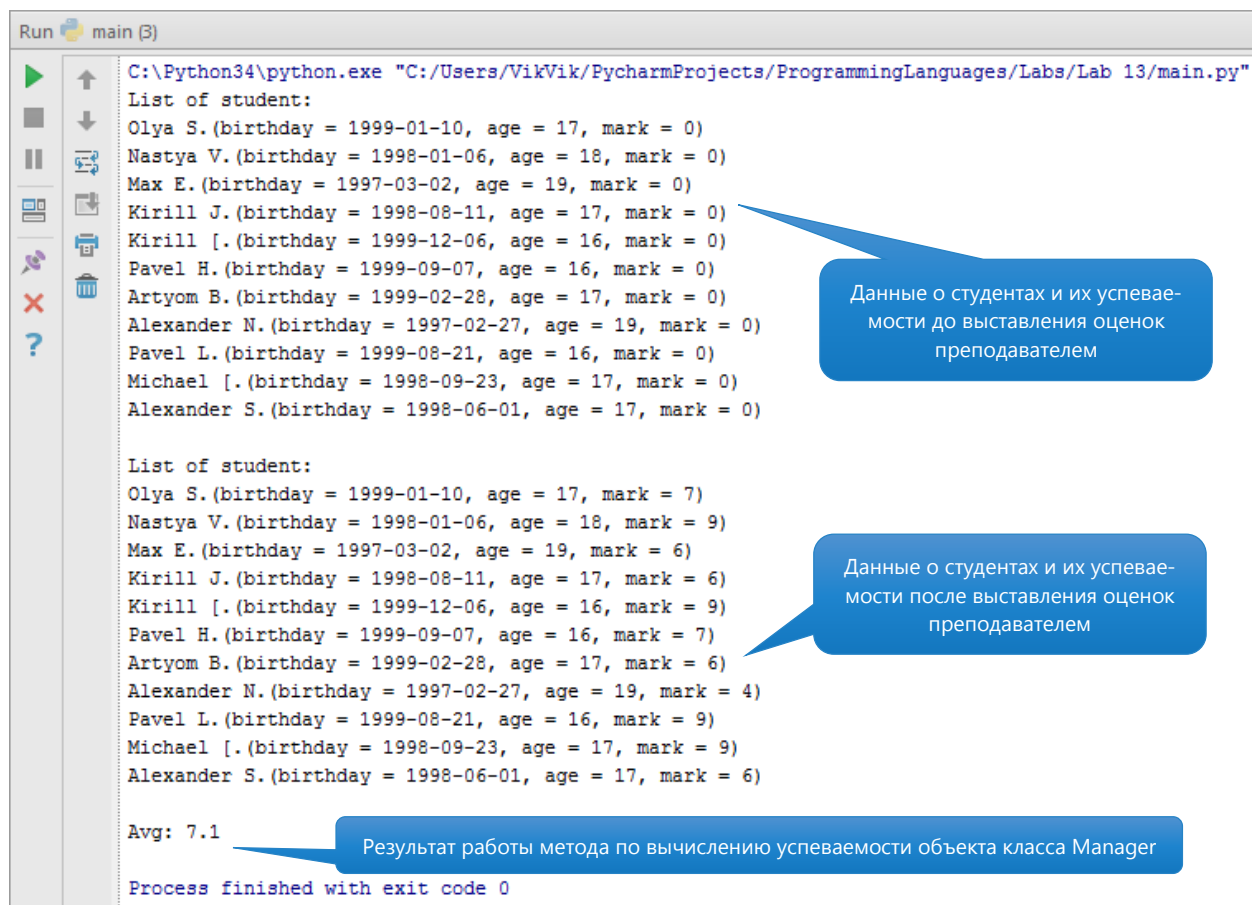


Рисунок 6 – Стартовый (тестовый) модуль main



```
Run main (3)
C:\Python34\python.exe "C:/Users/VikVik/PycharmProjects/ProgrammingLanguages/Labs/Lab 13/main.py"
List of student:
Olya S.(birthday = 1999-01-10, age = 17, mark = 0)
Nastya V.(birthday = 1998-01-06, age = 18, mark = 0)
Max E.(birthday = 1997-03-02, age = 19, mark = 0)
Kirill J.(birthday = 1998-08-11, age = 17, mark = 0)
Kirill I.(birthday = 1999-12-06, age = 16, mark = 0)
Pavel H.(birthday = 1999-09-07, age = 16, mark = 0)
Artyom B.(birthday = 1999-02-28, age = 17, mark = 0)
Alexander N.(birthday = 1997-02-27, age = 19, mark = 0)
Pavel L.(birthday = 1999-08-21, age = 16, mark = 0)
Michael I.(birthday = 1998-09-23, age = 17, mark = 0)
Alexander S.(birthday = 1998-06-01, age = 17, mark = 0)

List of student:
Olya S.(birthday = 1999-01-10, age = 17, mark = 7)
Nastya V.(birthday = 1998-01-06, age = 18, mark = 9)
Max E.(birthday = 1997-03-02, age = 19, mark = 6)
Kirill J.(birthday = 1998-08-11, age = 17, mark = 6)
Kirill I.(birthday = 1999-12-06, age = 16, mark = 9)
Pavel H.(birthday = 1999-09-07, age = 16, mark = 7)
Artyom B.(birthday = 1999-02-28, age = 17, mark = 6)
Alexander N.(birthday = 1997-02-27, age = 19, mark = 4)
Pavel L.(birthday = 1999-08-21, age = 16, mark = 9)
Michael I.(birthday = 1998-09-23, age = 17, mark = 9)
Alexander S.(birthday = 1998-06-01, age = 17, mark = 6)

Avg: 7.1
Process finished with exit code 0
```

Данные о студентах и их успеваемости до выставления оценок преподавателем

Данные о студентах и их успеваемости после выставления оценок преподавателем

Результат работы метода по вычислению успеваемости объекта класса Manager

Рисунок 7 – Результат тестирования работы программной системы «Университет»

Контрольные вопросы

1. Что такое ООП? Опишите базовые концепции, которые лежат в основе данной методологии программирования.
2. Для чего было создано ООП? На каких принципах базируется ООП? Опишите их основные идеи?
3. Приведите преимущества и недостатки объектно-ориентированного подхода.
4. Что такое объект и чем характеризуются объекты в ООП?
5. Что такое класс и зачем он нужен? Приведите общее определение класса в ООП. Какая разница между классом и объектом в ООП? Какие разновидности классов существуют в ООП?
6. Как описывается класс в Python? Каким способом можно создать объект (экземпляр класса) в Python?
7. Как в терминах ООП называется метод, который вызывается сразу после создания объекта для инициализации его состояния. Как его описать в классе?
8. Как можно обратиться к полям и методам объекта? В чём состоит особенность первого аргумента в методах класса?
9. Какой стандартный метод при описании класса необходимо переопределить, чтобы гарантировать, что в тех ситуациях, где требуется строковое представление объекта данного класса, система автоматических вызывала данный метод у ссылочной переменной?
10. Что такое UML? Как с помощью UML можно описать классы и их взаимосвязь друг с другом (т.е. описать UML-диаграмму классов)?

